# Stabilisateur électronique – Listing

Alexandre IOOSS

## Table des matières

# 1 Code utilisant le Framework Arduino (pour ATMEGA328p et ESP8266)

## 1.1 Bibliothèque pour asservir en position un moteur Brushless

Bibliothèque `BrushlessServo` maintenant publiée dans les contributions de l'éditeur Arduino.

### 1.1.1 BrushlessServo.h

```
#ifndef BRUSHLESS_SERVO_H
#define BRUSHLESS_SERVO_H

#include "Arduino.h"
#include "SinArray.h"

#ifndef PWMRANGE // PWMRANGE is already defined on some microcontroller
#define PWMRANGE 255 // Default for ATMEGA328 (Arduino UNO)
#endif

class BrushlessServo {
public:
  void attach(int p1, int p2, int p3); // Initialization
  void write(float angle);             // Move to a angle in degree
  void setOutputPower(int power_mult); // Set a power multiplier from 0 to PWMRANGE
  void setCycles(int n);               // Set how many sinusoide periods are
                                       // needed for one revolution
private:
  SinArray sinArray;
  int pins[3];
  int power = PWMRANGE;
  int n_cycles = 8;
};

#endif
```

### 1.1.2 BrushlessServo.cpp

```
#include "BrushlessServo.h"

void BrushlessServo::attach(int p1, int p2, int p3) {
  // Set output pins
  pins[0] = p1;
  pins[1] = p2;
  pins[2] = p3;
  pinMode(p1, OUTPUT);
  pinMode(p2, OUTPUT);
  pinMode(p3, OUTPUT);

  // Generate sin table for faster control
  sinArray.generate();
}

void BrushlessServo::write(float angle) {
  float real_angle = n_cycles * angle;

  int pwm1 = (float)power * (sinArray.getSinDegree(real_angle) + 1.) / 2.;
  int pwm2 = (float)power * (sinArray.getSinDegree(real_angle + 120) + 1.) / 2.;
  int pwm3 = (float)power * (sinArray.getSinDegree(real_angle + 240) + 1.) / 2.;

  // Set PWMs
  analogWrite(pins[0], pwm1);
  analogWrite(pins[1], pwm2);
  analogWrite(pins[2], pwm3);
}

void BrushlessServo::setOutputPower(int power_mult) { power = power_mult; }
```

```
30
31 void BrushlessServo::setCycles(int n) { n_cycles = n; }
```

### 1.1.3 SinArray.h

```
1  // This objet stores sin values in memory for faster calculations.
2
3  #ifndef SIN_ARRAY_H
4  #define SIN_ARRAY_H
5
6  #include "Arduino.h"
7
8  #define N_VALUES 512 // divisions of one period optimized for an Arduino UNO
9  #define STORE_TYPE int16_t
10 #define STORE_TYPE_RANGE 32767
11
12 class SinArray {
13 public:
14   void generate();               // Initialization
15   double getSinDegree(double a); // Get sin with a in degree
16   double getSin(double r);       // Get sin with r in radian
17
18 private:
19   double getSinByOffset(int offset); // Move to a offset between 0 and PRECISION
20   static int sin_array[N_VALUES];
21   static bool sin_array_empty;
22 };
23
24 #endif
```

### 1.1.4 SinArray.cpp

```
1  #include "Arduino.h"
2  #include "SinArray.h"
3
4  int SinArray::sin_array[N_VALUES];
5  bool SinArray::sin_array_empty = true;
6
7  void SinArray::generate() {
8    // Skip if already done
9    if (!sin_array_empty) { return; }
10
11   // Generate sin array
12   for (int i = 0; i < N_VALUES; i++) {
13     sin_array[i] = sin(2. * M_PI * i / (double)N_VALUES) * STORE_TYPE_RANGE;
14   }
15
16   sin_array_empty = false;  // Mark done
17 }
18
19 double SinArray::getSinByOffset(int offset) {
20   // Offset must be positive and between 0 and N_VALUES-1
21   offset = offset % N_VALUES;
22   if (offset < 0) { offset = N_VALUES + offset; }
23
24   return sin_array[offset] / (double)STORE_TYPE_RANGE;
25 }
26
27 double SinArray::getSinDegree(double a) {
28   int offset = (a * N_VALUES) / 360;
29   return getSinByOffset(offset);
30 }
31
32 double SinArray::getSin(double r) {
33   int offset = (r * N_VALUES) / (2. * M_PI);
34   return getSinByOffset(offset);
35 }
```

## 1.2 Bibliothèque pour utiliser l'algorithme constructeur de la centrale inertielle

Programme basé sur l'example du projet *The I2C Device Library*.

### 1.2.1 centrale_inertielle.h

```
#ifndef IMU_DMP_h
#define IMU_DMP_h

#define PIN_SDA A4
#define PIN_SCL A5
#define PIN_INTERRUPTION 2

// Paramètres de calibration
#define X_GYRO_OFFSET 99
#define Y_GYRO_OFFSET -1
#define Z_GYRO_OFFSET -4
#define X_ACCEL_OFFSET -2073
#define Y_ACCEL_OFFSET -1429
#define Z_ACCEL_OFFSET 1147

extern bool centrale_inertielle_initialisee; // vraie si succès d'initialisation
extern float centrale_inertielle_angles[3];  // lacet, roulis et tangage (rad)

extern void initialiser_centrale_inertielle();
extern void boucle_centrale_inertielle();

#endif
```

### 1.2.2 centrale_inertielle.cpp

```
#include "centrale_inertielle.h"
#include "Arduino.h"
#include "MPU6050_6Axis_MotionApps20.h"

// Variables définies dans l'en-tête
bool centrale_inertielle_initialisee = false;
float centrale_inertielle_angles[3];

// Variables globales
uint16_t taille_mesure;             // Taille d'une mesure
uint16_t taille_fifo;               // Taille de la file d'attente FIFO
volatile bool interruption = false; // Interruption envoyée par la centrale
MPU6050 centrale_inertielle;

// Fonction d'interruption
void interruptionVraie() { interruption = true; }

// Initialise la centrale inertielle avec l'algorithme de MotionApps
void initialiser_centrale_inertielle() {
  #if defined(__AVR_ATmega328P__)
  Wire.begin(); // I2C ATMEGA
  #else
  Wire.begin(PIN_SDA, PIN_SCL); // I2C ESP8266
  #endif
  Wire.setClock(400000);

  pinMode(PIN_INTERRUPTION, INPUT); // Entrée de l'interruption

  centrale_inertielle.initialize(); // Initialisation de la centrale
  uint8_t etat = centrale_inertielle.dmpInitialize(); // Initialisation DMP

  // Calibration
  centrale_inertielle.setXGyroOffset(X_GYRO_OFFSET);
  centrale_inertielle.setYGyroOffset(Y_GYRO_OFFSET);
  centrale_inertielle.setZGyroOffset(Z_GYRO_OFFSET);
```

```
  centrale_inertielle.setXAccelOffset(X_ACCEL_OFFSET);
  centrale_inertielle.setYAccelOffset(Y_ACCEL_OFFSET);
  centrale_inertielle.setZAccelOffset(Z_ACCEL_OFFSET);

  if (etat == 0) {
    // La centrale marche
    centrale_inertielle.setDMPEnabled(true); // Active le module de calcul
    attachInterrupt(digitalPinToInterrupt(PIN_INTERRUPTION), interruptionVraie,
                    RISING); // Interruption envoyée par la centrale

    // Récupération de la taille d'une mesure
    taille_mesure = centrale_inertielle.dmpGetFIFOPacketSize();

    centrale_inertielle_initialisee = true;
  }
}

// Récupère des nouvelles mesures si elles sont disponibles
void boucle_centrale_inertielle() {
  if ((!interruption && taille_fifo < taille_mesure) ||
      !centrale_inertielle_initialisee) {
    return; // Pas encore prête
  }

  interruption = false;

  // Etat actuel de la centrale
  uint8_t etat_interruption = centrale_inertielle.getIntStatus();

  // Récupère la taille de la liste d'attente
  taille_fifo = centrale_inertielle.getFIFOCount();

  if ((etat_interruption & 0x10) || taille_fifo == 1024) {
    // Dépassement de capacité, on réinitialise
    centrale_inertielle.resetFIFO();
  } else if (etat_interruption & 0x02) {
    // Attente d'avoir toutes les données (très rapide)
    while (taille_fifo < taille_mesure) {
      taille_fifo = centrale_inertielle.getFIFOCount();
    }

    // Lecture de la liste FIFO
    uint8_t fifoBuffer[64];
    centrale_inertielle.getFIFOBytes(fifoBuffer, taille_mesure);

    // Nouvelle taille de la liste d'attente
    taille_fifo -= taille_mesure;

    // Calcul de l'angle en passant par les quaternions
    Quaternion q;  // Représente l'orientation
    VectorFloat p; // Vecteur pesanteur
    centrale_inertielle.dmpGetQuaternion(&q, fifoBuffer);
    centrale_inertielle.dmpGetGravity(&p, &q);
    centrale_inertielle.dmpGetYawPitchRoll(centrale_inertielle_angles, &q, &p);
  }
}
```

## 1.3 Programme pour envoyer des commandes en rampe au servomoteur (ESP8266)

### 1.3.1 main.cpp

```
#include "Arduino.h"
#include "ESP8266WiFi.h"
#include <Hash.h>
#include <Servo.h>
#include <SimpleTimer.h>
#include <WebSocketsServer.h>

// Timers
SimpleTimer timer_calcul_commande;
SimpleTimer timer_transmission_websocket;

// Objets
Servo servomoteur;
WebSocketsServer webSocket = WebSocketsServer(81);

// Variables pour la commande
unsigned long temps_commande = 0; // ms, temps à partir du début de la commande
unsigned long temps_commande_orig = 0; // ms, décalage temporel de l'origine
unsigned long temps_duree_rampe = 0;   // ms, durée de la rampe
int angle_servo = 950;                 // us, angle à envoyer au servomoteur
const int angle_servo_min = 950;       // us, angle minimal du servomoteur
const int angle_servo_max = 1800;      // us, angle maximal du servomoteur

// Initialisation du point d'accès Wifi
void initialiser_wifi() {
  WiFi.mode(WIFI_AP);                 // Point d'accès WiFi
  IPAddress ip_stab(192, 168, 4, 1);  // IP du stabilisateur & passerelle
  IPAddress masque(255, 255, 255, 0); // Masque de sous-réseau
  WiFi.softAPConfig(ip_stab, ip_stab, masque); // Serveur DHCP
  WiFi.softAP("Stabilisateur", "t1p3stabi");   // Nom et mot de passe du Wifi
}

// Commande en rampe
void commande_servomoteur() {
  if (temps_duree_rampe == 0) {
    // Aucune donnée n'a été reçue donc retour à l'origine
    angle_servo = angle_servo_min;
  } else {
    temps_commande = millis() - temps_commande_orig;

    if (temps_commande > 5000 and temps_commande < temps_duree_rampe + 5000) {
      // Effectue la rampe
      angle_servo = (temps_commande - 5000) *
                        (angle_servo_max - angle_servo_min) /
                        temps_duree_rampe +
                    angle_servo_min;
    } else if (temps_commande > temps_duree_rampe + 10000) {
      // La rampe est terminéee
      temps_duree_rampe = 0;
    }
  }

  servomoteur.writeMicroseconds(angle_servo);
}

// Transmet une mesure vers le socket réseau
void transmission_websocket() {
  int pot = analogRead(A0);
  String texte =
      String(temps_commande) + "," + String(angle_servo) + "," + String(pot);
  webSocket.broadcastTXT(texte);
}

```

```
// Gère la réception d'une commande (inspiré d'un exemple de WebSocket)
void rx_websocket(uint8_t num, WStype_t type, uint8_t *payload, size_t length) {
  if (type == WStype_TEXT and payload[0] == '#') {
    // Parsing de la commande (de la forme #FFF)
    uint32_t recu = (uint32_t)strtol((const char *)&payload[1], NULL, 16);
    int num_vitesse = ((recu >> 4) & 0xFF);      // 2 premiers chiffres
    int exp_vitesse = ((recu >> 0) & 0xF) - 0x8; // Puissance de 10

    // Calcul de la vitesse et de la durée de la rampe
    float vitesse = num_vitesse * pow(10, exp_vitesse);                // us/ms
    temps_duree_rampe = (angle_servo_max - angle_servo_min) / vitesse; // ms

    // Déplacement de l'origine temporel
    temps_commande_orig = millis();

    // Renvoie de la durée pour vérification
    webSocket.broadcastTXT("#" + String(temps_duree_rampe));
  }
}

// Initialisation
void setup() {
  servomoteur.attach(D6);          // Commande du servomoteur
  pinMode(A0, INPUT);              // Entrée du potentiomètre du servomoteur
  initialiser_wifi();              // Point d'accès Wifi
  webSocket.begin();               // Initialisation du Websocket
  webSocket.onEvent(rx_websocket); // Réception d'une commande

  timer_calcul_commande.setInterval(25, commande_servomoteur);
  timer_transmission_websocket.setInterval(75, transmission_websocket);
}

// Boucle principale qui lance les sous-boucles
void loop() {
  timer_calcul_commande.run();
  timer_transmission_websocket.run();
  webSocket.loop();
}
```

## 1.4 Programme embarqué sur le stabilisateur finalisé (ESP8266)

### 1.4.1 main.cpp

```
#include "Arduino.h"
#include "ESP8266WiFi.h"
#include "centrale_inertielle.h"
#include <Hash.h>
#include <PID_v1.h>
#include <Servo.h>
#include <SimpleTimer.h>
#include <WebSocketsServer.h>

// Variables pour la commande
double angle_mesure = 0.; // rad, angle mesuré par la centrale inertielle
double angle_servo = 0.;  // us, angle à envoyer au servomoteur
double angle_voulu = 0.;  // Angle correspondant à l'horizontale

// Objets
SimpleTimer timer_transmission_websocket;
Servo servomoteur;
WebSocketsServer webSocket = WebSocketsServer(81);
PID correcteur(&angle_mesure, &angle_servo, &angle_voulu, 100, 3300, 0,
               REVERSE); // Kp=100, Ki=Kd=0

// Initialisation du point d'accès Wifi
void initialiser_wifi() {
  WiFi.mode(WIFI_AP);                 // Point d'accès WiFi
  IPAddress ip_stab(192, 168, 4, 1);  // IP du stabilisateur & passerelle
  IPAddress masque(255, 255, 255, 0); // Masque de sous-réseau
  WiFi.softAPConfig(ip_stab, ip_stab, masque); // Serveur DHCP
  WiFi.softAP("Stabilisateur", "t1p3stabi");   // Nom et mot de passe du Wifi
}

// Transmet une mesure vers le socket réseau
void transmission_websocket() {
  int pot = analogRead(A0);
  int temps_actuel = millis();
  String texte = String(temps_actuel) + "," + String(angle_servo) + "," +
                 String(pot) + "," + String(angle_mesure);
  webSocket.broadcastTXT(texte);
}

// Gère la réception d'une commande
void rx_websocket(uint8_t num, WStype_t type, uint8_t *payload, size_t length) {
  if (type == WStype_BIN) {
    double *donnees = (double *)payload; // On reçoit 3 double

    if (length / sizeof(double) != 3) {
      // Si on ne reçoit pas 3 éléments, alors problème
      return;
    }

    // On change les paramètres du correcteur
    correcteur.SetTunings(donnees[0], donnees[1], donnees[2]);
  }
}

// Initialisation
void setup() {
  servomoteur.attach(D6);            // Commande du servomoteur
  pinMode(A0, INPUT);                // Entrée du potentiomètre du servomoteur
  initialiser_centrale_inertielle(); // Centrale inertielle
  initialiser_wifi();                // Point d'accès Wifi
  webSocket.begin();                 // Initialisation du Websocket
  webSocket.onEvent(rx_websocket);   // Réception d'une commande

```

```
64    // Transmet une valeur toutes les 60ms
65    timer_transmission_websocket.setInterval(60, transmission_websocket);
66
67    correcteur.SetOutputLimits(-825, 825); // Domaine délimité en us
68    correcteur.SetSampleTime(20);          // Calcul tous les 25ms
69    correcteur.SetMode(AUTOMATIC);         // Active le correcteur
70  }
71
72  // Boucle principale qui lance les sous-boucles
73  void loop() {
74    // Appel des différents Timer
75    boucle_centrale_inertielle();       // Centrale inertielle
76    correcteur.Compute();               // Correcteur
77    timer_transmission_websocket.run(); // Transmission des données
78    webSocket.loop();                   // WebSocket
79
80    // Transfert des variables
81    angle_mesure = centrale_inertielle_angles[2];
82    servomoteur.writeMicroseconds(angle_servo+1375);
83  }
```

## 1.5 Programme embarqué sur le banc d'essais (ATMega328p)

### 1.5.1 main.cpp

```
#include "Arduino.h"
#include "BrushlessServo.h"
#include "SinArray.h"
#include "centrale_inertielle.h"

// Objets
BrushlessServo moteur;
SinArray sinArray;

// Variables pour la commande
int amplitude = 10; // en degrés
float pulsation = 0.;
float angle_moteur = 0.;        // en degrés
unsigned long temps_debut = 0; // origine temporelle

unsigned long temps_precedent_commande_moteur_boucle = 0;
unsigned long temps_precedent_transmission_serie_boucle = 0;
unsigned long temps_actuel = 0; // en ms

// Commande du moteur
void commande_moteur() {
  // Toutes les 10ms
  if (temps_actuel - temps_precedent_commande_moteur_boucle >= 10) {
    temps_precedent_commande_moteur_boucle = temps_actuel;

    // Commande du moteur
    if (pulsation > 0.1) {
      angle_moteur =
          amplitude * sinArray.getSin(pulsation * temps_actuel / 1000.);
      moteur.write(-angle_moteur - 20);
    }
  }
}

// Gère la réception d'une commande
void reception_serie() {
  if (Serial.available() > 0) {      // Si on a reçu quelque chose
    pulsation = Serial.parseFloat(); // Reception d'une nouvelle pulsation

    // Nouvelle origine des temps
    temps_debut = millis();
    temps_precedent_commande_moteur_boucle = 0;
    temps_precedent_transmission_serie_boucle = 0;
  }
}

// Transmet une mesure sur le port série
void transmission_serie() {
  // Toutes les 10 ms
  if (temps_actuel - temps_precedent_transmission_serie_boucle >= 10) {
    temps_precedent_transmission_serie_boucle = temps_actuel;

    Serial.print(millis() - temps_debut);
    Serial.print(",");
    Serial.print(String(angle_moteur, 4));
    Serial.print(",");
    Serial.print(analogRead(A0));
    Serial.print(",");
    Serial.print(String(centrale_inertielle_angles[0], 4));
    Serial.print(",");
    Serial.print(String(centrale_inertielle_angles[1], 4));
    Serial.print(",");
    Serial.println(String(centrale_inertielle_angles[2], 4));
```

```
  }
}

// Initialisation
void setup() {
  // Fast PWM Mode, diviseur de 1 : 16MHz/256
  // Extrait de https://www.arduino.cc/en/Tutorial/SecretsOfArduinoPWM
  TCCR1B = (TCCR1B & 0b11111000) | 0x01;
  TCCR2B = (TCCR2B & 0b11111000) | 0x01;

  moteur.attach(9, 10, 11); // Définition du moteur Brushless
  // moteur.setOutputPower(160);  // Réduction de la tension

  Serial.begin(115200);              // Communication série
  pinMode(A0, INPUT);                // Entrée du potentiomètre
  sinArray.generate();               // Précalcul du table de valeurs de sinus
  initialiser_centrale_inertielle(); // Centrale inertielle
}

// Boucle principale du programme
void loop() {
  // On arrête le programme si la centrale ne fonctionne pas
  if (!centrale_inertielle_initialisee)
    return;

  temps_actuel = millis() - temps_debut;

  boucle_centrale_inertielle(); // Centrale inertielle
  commande_moteur();            // Commande du moteur
  transmission_serie();         // Transmission des mesures
  reception_serie();            // Réception d'une commande
}
```

## 1.6 Bibliothèques externes

### 1.6.1 WebSockets (Arduino Framework)

Utilisée sur le stabilisateur pour héberger un serveur au standard WebSocket.

Sous licence GNU LGPL 2.1+.

### 1.6.2 SimpleTimer (Arduino Framework)

Utilisée sur le stabilisateur pour obtenir un semblant de multitâche.

Sous licence GNU LGPL 2.1+.

### 1.6.3 PID (Brett Beauregard, Arduino Framework)

Utilisée sur le stabilisateur pour l'asservissement.

Développée par *Brett Beauregard* sous licence MIT.

### 1.6.4 I2Cdevlib-MPU6050 (The I2C Device Library)

Utilisée pour envoyer l'algorithme constructeur (de MotionApps) sur le Digital Motion Unit de la centrale inertielle.

Dérivée par le projet *The I2C Device Library* de la librarie de *MotionApps* sous licence MIT.

# 2 Code Python

## 2.1 Script pour tester le stabilisateur et configurer les coefficients du correcteur à la volée

### 2.1.1 test_stabilisateur.py

```python
from websocket import create_connection
import matplotlib.pyplot as plt
import numpy as np
from struct import pack


def faire_acquisition(kp, ki, kd, n_mesures):
    """ Règle le correcteur et réalise une acquisition en Wifi """

    ws = create_connection("ws://192.168.4.1:81")

    # Réglage du correcteur
    param_bin = pack('ddd', kp, ki, kd)
    ws.send_binary(param_bin)

    # Réception
    donnees = np.zeros((n_mesures, 4), dtype=np.float)
    for i in range(n_mesures):
        recu = ws.recv()
        print(recu)
        donnees[i, :] = recu.split(",")

    ws.close()

    # Conversions
    temps = (donnees[:, 0] - donnees[0, 0]) / 1000  # Temps en seconde
    servo = donnees[:, 1] + 1375  # Commande servomoteur en us
    pot = donnees[:, 2]  # Valeur CAN du potentiomètre
    imu = donnees[:, 3] * 180 / np.pi  # Angle de la centrale en degrées

    return temps, servo, pot, imu


def tracer_entree_sortie_correcteur(temps, servo, pot, imu):
    plt.figure(figsize=(12, 4))

    plt.subplot("121")
    plt.plot(temps, imu, label="Centrale inertielle")
    plt.title("Signal d'erreur")
    plt.xlabel("Temps (secondes)")
    plt.ylabel("Angle (degrés)")
    plt.legend(); plt.grid(); plt.ylim(-90, 90)

    plt.subplot("122")
    plt.plot(temps, servo, label="Commande servomoteur")
    plt.title("Sortie du correcteur")
    plt.xlabel("Temps (secondes)")
    plt.ylabel("Commande (us)")
    plt.legend(); plt.grid(); plt.ylim(550, 2200)

    plt.tight_layout()
    plt.show()


# Réglages
kp, ki, kd = 100., 3300., 0.
n_mesures = 100

temps, servo, pot, imu = faire_acquisition(kp, ki, kd, n_mesures)
tracer_entree_sortie_correcteur(temps, servo, pot, imu)
```

## 2.2 Script pour commander le banc d'essais

### 2.2.1 ComBanc.py : définit un objet représentant le banc d'essais

```python
import serial
import time
import numpy as np


class ComBanc(serial.Serial):
    """
    Représente la communication avec le banc d'essais, hérite de Serial
    """

    def __init__(self, baudrate=9600, port='COM1'):
        super().__init__()

        self.baudrate = baudrate
        self.port = port

    def open(self):
        # Ouverture de la communication
        super().open()

        # Attendre que le banc démarre
        time.sleep(2)

    def changer_pulsation(self, omega):
        """ Permet de changer la pulsation de la sollicitation harmonique """

        print("Nouvelle pulsation :", omega)

        # Envoie de la commande
        cmd = str(omega)
        self.write(cmd.encode('utf-8'))

        # Attendre que le banc prenne la valeur en compte
        time.sleep(2)

    def acquisition(self, n):
        """ Permet d'acquerir n valeurs séparées par une virgule """

        # Vider le buffer d'entree
        self.flushInput()

        # Information
        print("Acquisition de " + str(n) + " couples")

        # Acquisition
        mesures = []
        for i in range(n):
            try:
                rx = str(self.readline()).split(",")
                mesures.append([
                    float(rx[0][2:]),
                    float(rx[1]),
                    float(rx[2]),
                    float(rx[3]),
                    float(rx[4]),
                    float(rx[5][:-5])
                ])
                print(mesures[-1])
            except ValueError:
                print("Erreur de reception")

        return np.array(mesures)
```

### 2.2.2 Bode.py : définit un objet représentant un diagramme de Bode

```python
import numpy as np
import matplotlib.pyplot as plt


class Bode:
    """
    Objet représentant un diagramme de Bode
    """

    def __init__(self, title="", puissance_dix_min=-1, puissance_dix_max=2):
        self.title = title
        self.puissance_dix_min = puissance_dix_min
        self.puissance_dix_max = puissance_dix_max

        self.omegas = []
        self.gains = []
        self.phases = []

    def plot_gain(self):
        """ Trace le diagramme des gains """

        plt.plot(self.omegas, self.gains, color="green")

        # Réglages
        plt.xscale("log")
        plt.xlim(10**self.puissance_dix_min, 10**self.puissance_dix_max)
        plt.title(self.title)
        plt.grid()
        plt.xlabel("x")
        plt.ylabel("GdB")
        plt.axhline(color="black")

    def plot_phase(self):
        """ Trace le diagramme des phases """

        plt.plot(self.omegas, self.phases, color="green")

        # Réglages
        plt.xscale("log")
        plt.xlim(10**self.puissance_dix_min, 10**self.puissance_dix_max)
        plt.grid()
        plt.xlabel("omega")
        plt.ylabel("Phase")
        plt.axhline(color="black")

    def plot(self):
        """ Trace le diagramme de Bode complet """

        plt.subplot(211)
        self.plot_gain()

        plt.subplot(212)
        self.plot_phase()
```

### 2.2.3 BodeTF.py : extension de l'objet Bode pour tracer une fonction de transfert

```python
import numpy as np
import matplotlib.pyplot as plt
from Bode import Bode


class BodeTF(Bode):
    """
    Objet représentant un diagramme de Bode
    créé à partie d'une fonction de transfert
    """

    def calc_gain(v):
        """ Calcule le gain en décibel du complexe v """

        return 20 * np.log10(abs(v))

    def calc_phase(v):
        """ Calcule la phase du complexe v """

        r = v.real
        i = v.imag

        if r == 0:
            return 0
        elif r * i > 0:
            return np.arctan(i / r) - np.pi
        else:
            return np.arctan(i / r)

    def plot(self, fonction_de_transfert):
        """ Trace le diagramme des gains et des phases d'une fonction de transfert """

        # Calcul des points
        self.omegas = np.logspace(self.ten_pow_min, self.ten_pow_max, 100)
        vals = [fonction_de_transfert(x) for x in self.omegas]
        self.gains = [self.calc_gain(v) for v in vals]
        self.phases = [self.calc_phase(v) for v in vals]

        # Affichage
        super().plot()
```

### 2.2.4 test_banc.py : exemple de code pour une commande du banc

```python
from ComBanc import ComBanc
import matplotlib.pyplot as plt
import numpy as np
import time

amplitude = 10


def acquisition(banc, omega):
    """
    Réalise une sollicitation harmonique et renvoie les mesures du potentiomètre
    et de la centrale inertielle en fonction du temps ainsi que la commande
    """

    banc.open()  # Ouvre la connexion et démarre le banc d'essais
    time.sleep(5)  # Laisse le temps à la centrale de converger
    banc.changer_pulsation(omega)

    # Acquisition de 1000 valeurs (1000*0.01 = 1s)
    mesures = banc.acquisition(1000)

    # Alignement du potentiomètre et conversions
    temps = mesures[:, 0] / 1000  # en secondes
    max_pot = max(mesures[:, 2])
    moy_pot = (max_pot + min(mesures[:, 2])) / 2
    commande = mesures[:, 1]
    potentiometre = (mesures[:, 2] - moy_pot) / (
        max_pot - moy_pot) * amplitude  # en degrées
    imu = -mesures[:, 5] * 180 / np.pi  # en degrées

    banc.close()  # Fermeture de la connexion

    return [temps, commande, potentiometre, imu]


def tracer_potentiometre_imu(temps, commande, potentiometre, imu):
    plt.figure(figsize=(8, 4))

    plt.subplot(211)
    plt.title("Commande et retour potentiomètre")
    plt.plot(temps, commande, "+:", label="Commande")
    plt.plot(temps, potentiometre, "+:", label="Potentiomètre")
    plt.xlabel("Temps (en secondes)")
    plt.grid()
    plt.legend()

    plt.subplot(212)
    plt.title("Commande et retour MPU6050")
    plt.plot(temps, commande, "+:", label="Commande")
    plt.plot(temps, imu, "+:", label="MPU6050 (Roulis)")
    plt.xlabel("Temps (en secondes)")
    plt.grid()
    plt.legend()

    plt.tight_layout()


# Instance du banc d'essais
banc = ComBanc(baudrate=115200, port='/dev/ttyUSB0')

temps, commande, potentiometre, imu = acquisition(banc, 3.14156)
tracer_potentiometre_imu(temps, commande, potentiometre, imu)

plt.show()
```

### 2.2.5 bode_centrale_inertielle.py : Création du Bode expérimental de la centrale inertielle

```python
from Bode import Bode
from ComBanc import ComBanc
import matplotlib.pyplot as plt
import numpy as np
import time


def acquisition(banc, omega):
    """
    Réalise une sollicitation harmonique et renvoie les mesures du potentiomètre
    et de la centrale inertielle en fonction du temps ainsi que la commande
    """

    banc.open()  # Ouvre la connexion et démarre le banc d'essais
    time.sleep(5)  # Laisse le temps à la centrale de converger
    banc.changer_pulsation(omega)

    # Acquisition de 1000 valeurs (1000*0.01 = 1s)
    mesures = banc.acquisition(1000)

    # Alignement du potentiomètre et conversions
    temps = mesures[:, 0] / 1000  # en secondes
    max_pot = max(mesures[:, 2])
    moy_pot = (max_pot + min(mesures[:, 2])) / 2
    commande = mesures[:, 1]
    potentiometre = (mesures[:, 2] - moy_pot) / (
        max_pot - moy_pot) * 10  # en degrées
    imu = -mesures[:, 5] * 180 / np.pi  # en degrées

    banc.close()  # Fermeture de la connexion

    return [temps, commande, potentiometre, imu]


def gain(entree, sortie):
    amplitude_entree = max(entree) - min(entree)
    amplitude_sortie = max(sortie) - min(sortie)

    return 20 * np.log10(abs(amplitude_sortie / amplitude_entree))


# Instance du banc d'essais
banc = ComBanc(baudrate=115200, port='/dev/ttyUSB0')
""" POUR FAIRE L'ACQUISITION D'UN POINT
temps, commande, potentiometre, imu = acquisition(banc, 1.5)
print(gain(potentiometre[5:], imu[5:]))

plt.figure(figsize=(8, 4))
plt.plot(temps[5:], potentiometre[5:], "+:", label="Potentiomètre")
plt.plot(temps[5:], imu[5:], "+:", label="IMU")
plt.xlabel("Temps (en secondes)")
plt.grid(); plt.legend(); plt.show()
"""

# Trace le diagramme de Bode
bode_imu = Bode(title="Gain de Bode expérimental de la centrale inertielle")

bode_imu.omegas, bode_imu.gains = np.loadtxt("bode_imu.csv")

# On affiche le résultat
plt.figure(figsize=(8, 4))
bode_imu.plot_gain()
plt.tight_layout()
plt.show()
```

## 2.3 Script pour le traitement du signal du pendule pesant

### 2.3.1 pendule.py

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats


def lireDonneesLatis(nom_fichier):
    """ Charge le fichier Latis en une matrice de données """

    donnees = np.loadtxt(nom_fichier, delimiter=';', skiprows=1)
    donnees[:, 1] *= 9  # conversion tension en degree, facteur constructeur
    return donnees


def trouverCoupures(liste_grandeur, liste_temps):
    """ Renvoie les temps où la grandeur passe du signe - à + """

    liste_coupures = []
    for i, t in enumerate(liste_temps[:-1]):
        if liste_grandeur[i] < 0 and liste_grandeur[i + 1] > 0:
            liste_coupures.append([i, t])
    return np.array(liste_coupures)


def tronquerDonnees(donnees, n_periodes):
    """ Tronque le début des données et laisse n_periodes périodes """

    liste_coupures = trouverCoupures(donnees[:, 1], donnees[:, 0])
    debut = int(liste_coupures[0, 0])
    if n_periodes:
        fin = int(liste_coupures[n_periodes, 0])
    else:
        fin = -1
    return donnees[debut:fin, :], liste_coupures[:n_periodes, :]


def plotAngle(titre, fichier, from_i, n_periodes, calcul_periodes=True):
    """ Trace l'angle du pendule à partir du point from_i sur n_periodes périodes """

    donnees_brut = lireDonneesLatis(fichier)

    # Centre les points
    donnees_brut[:, 1] += -np.average(donnees_brut[-10000:, 1])

    # Tronque les données
    donnees, liste_coupures = tronquerDonnees(donnees_brut[from_i:, :],
                                              n_periodes)
    plt.xlim(donnees[0, 0], donnees[-1, 0])
    m = max(donnees[:, 1]) + 1
    plt.ylim(-m, m)

    # Graduations et calcul de la période
    if calcul_periodes:
        plt.gca().set_xticks(np.round(liste_coupures[:, 1], 1))
        O = []
        for i in range(len(liste_coupures[:, 1]) - 1):
            O.append(liste_coupures[i + 1, 1] - liste_coupures[i, 1])
        moyenne = np.average(O)
        ecart_type = np.std(O)
        periode = "Période = " + str(np.round(moyenne, 2)) + " +/- " + str(
            np.round(ecart_type, 2)) + " s"
        style_text_pyplot = dict(
            boxstyle='square', facecolor='white', alpha=0.8)
        plt.text(
```

```
            donnees[-1, 0],
            m + 1,
            titre + " : " + periode,
            bbox=style_text_pyplot,
            fontsize=12,
            verticalalignment='top',
            horizontalalignment='right')

    # Graphique
    plt.xlabel("Temps (en seconde)")
    plt.ylabel("Angle du\npendule (degrés)")
    plt.plot(donnees[:, 0], donnees[:, 1], "k.", markersize=1.)
    plt.grid()


for a in ["20", "30"]:
    plt.figure(figsize=(10, 3))
    plt.subplot(211)
    plotAngle("Sans la caméra", "s_" + a + ".csv", -6000, 20)
    plt.subplot(212)
    plotAngle("Avec la caméra", "a_" + a + ".csv", -6000, 20)
    plt.tight_layout()

plt.show()
```

# 3 Code MATLAB pour acquérir les rampes envoyées au servomoteur

## 3.1 Fonctions

### 3.1.1 reel_faireAcquisition.m : acquisition d'une rampe sur le servomoteur

```
function [ commandes, reponses ] = reel_faireAcquisition( vitesse )
% Permet d'effectuer l'acquisition d'une rampe de vitesse (en us/ms)

sys = reel_SystemeReel();                  % Connexion au système réel
sys.nouvelle_rampe(vitesse)                % Lance une rampe à la vitesse donnée en us/s
waitfor(sys, 'faire_acquisition', false)   % Attend la fin de l'acquisition
donnees_reel = sys.donnees;                % Récupère les données
sys.close();                               % Ferme la connexion

% Extrait les colonnes
commandes = [donnees_reel(:,1)/1000, donnees_reel(:,2)];
reponses = [donnees_reel(:,1)/1000, donnees_reel(:,3)];

end
```

### 3.1.2 reel_faireRegression.m : régression affine entre l'entrée et la sortie

```
function [ coeff_regression, rcarre ] = reel_faireRegression( commande, reponse )
% Donne les coefficients de la fonction affine
% permettant de passer du retour du potentiomètre à la commande en us

coeff_regression = polyfit(commande, reponse, 1);  % Regression

% Calcul du coeff r au carré
yfit = polyval(coeff_regression, commande);
yresid = reponse - yfit;
SStotal = (length(reponse)-1) * var(reponse);
rcarre = 1 - sum(yresid.^2)/SStotal;

end
```

### 3.1.3 reel_rognerRampe.m : rogne le signal de 5s au début et à la fin

```
function [ temps_rognee, commande_rognee, reponse_rognee ] = reel_rognerRampe( temps,
    commande, reponse )
% Supprime les 5 premières et dernières secondes pour ne garder que la rampe

reel_periode_echantillonage = 100; % ms
t_rognage = 5000; % ms
n = floor(t_rognage/reel_periode_echantillonage);

temps_rognee = temps(n:end-n);
commande_rognee = commande(n:end-n);
reponse_rognee = reponse(n:end-n);

end
```

### 3.1.4 reel_traceCourbe.m : trace l'entrée et la sortie du servomoteur

```
function reel_traceCourbe( entree, sortie )
% Trace l'entrée et la sortie selon le temps.

figure; hold all; grid on;
plot(entree, 'LineWidth', 2);
plot(sortie, 'LineWidth', 2);
legend('Commande de position (us)', 'Position mesurée (us)')
legend('Location', 'southeast')
xlabel('t (en secondes)')

end
```

### 3.1.5 reel_traceRegression.m : trace la sortie en fonction de l'entrée du servomoteur (avec superposition de la régression)

```
function reel_traceRegression( commande, reponse, coeff_regression, rcarre )
% Trace le signal du potentiomètre (10 bits) en fonction de la commande en us
% Ajoute par dessus la regression effectuée.

figure;
hold all;
grid on;
z = coeff_regression(1)*commande + coeff_regression(2);
plot(commande, [reponse z], 'LineWidth', 2);
legend('Signal expérimental', strcat('Réduction linéaire : r=', num2str(rcarre)),
    'Location', 'southeast')
xlabel('Commande')
ylabel('Réponse réelle filtrée')

end
```

### 3.1.6 reel_SystemeReel.m : définit un objet représentant la communication avec le système réel

```
classdef reel_SystemeReel < WebSocketClient
    % Abstraction de la communication avec le stabilisateur
    % Utilise la librarie MatlabWebSocket de Jérémy Béjanin (licence MIT)

    properties
        donnees  % Colonnes : temps, commande, potentiomètre
        faire_acquisition = false
        retour_commande_recu = false
        duree = 0
        nb_valeurs = 0
        i_valeur = 1
    end

    methods
        function obj = reel_SystemeReel()
            obj@WebSocketClient('ws://192.168.4.1:81');  % Objet parent
        end

        function lance_acquisition(obj)
            % Prépare le tableau en mémoire et lance l'acquisition
            obj.donnees = zeros(obj.nb_valeurs, 3);
            obj.i_valeur = 1;
            obj.faire_acquisition = true;
        end

        function nouvelle_rampe(obj, vitesse)
            % Demande au système d'effectuer une rampe

            vitesse_puissance_10 = floor(log10(vitesse))-1;
            vitesse_deux_chiffres = vitesse * 10^(-vitesse_puissance_10);

            message = strcat('#', sprintf('%02X', vitesse_deux_chiffres), sprintf('%X',
                vitesse_puissance_10+8));
            fprintf('Envoie la commande %s\n', message);
            obj.send(message);

            % Attend le retour de la commande
            waitfor(obj, 'retour_commande_recu', true);
            obj.retour_commande_recu = false;

            % Lance l'acquisition
            obj.lance_acquisition();
        end
    end

    methods (Access = protected)
```

```
        function onOpen(~, message)
            fprintf('%s\n',message);  % Message de connexion
        end

        function onTextMessage(obj, message)
            if obj.retour_commande_recu == false && message(1) == '#'
                % Si message commence par "#"
                % alors c'est un retour de commande donnant la durée
                fprintf('Retour : %s\n', message);
                len = length(message);
                obj.duree = str2double(message(2:len))+10000; % marge de 10s
                obj.nb_valeurs = uint32(obj.duree/100); % échantillon tous les 100ms
                obj.retour_commande_recu = true;
            elseif obj.faire_acquisition == true
                % Stocke les données si acquisition
                ligne = cellfun(@str2double, strsplit(message, ','));
                obj.donnees(obj.i_valeur, :) = ligne;

                % S'il y a eu assez d'acquisitions, alors on stoppe
                obj.i_valeur = obj.i_valeur + 1;
                if obj.i_valeur > obj.nb_valeurs
                    fprintf('Fin acquisition\n');
                    obj.faire_acquisition = false;
                end
            end
        end

        function onBinaryMessage(~, ~)
            % Ignorer
        end

        function onError(~, message)
            fprintf('Erreur : %s', message);  % Affiche l'erreur
        end

        function onClose(~, message)
            fprintf('%s\n', message);  % Message de déconnexion
        end
    end
end
```

## 3.2 Scripts d'exemple

### 3.2.1 Effectue un grand nombre d'essais pour affiner la régression

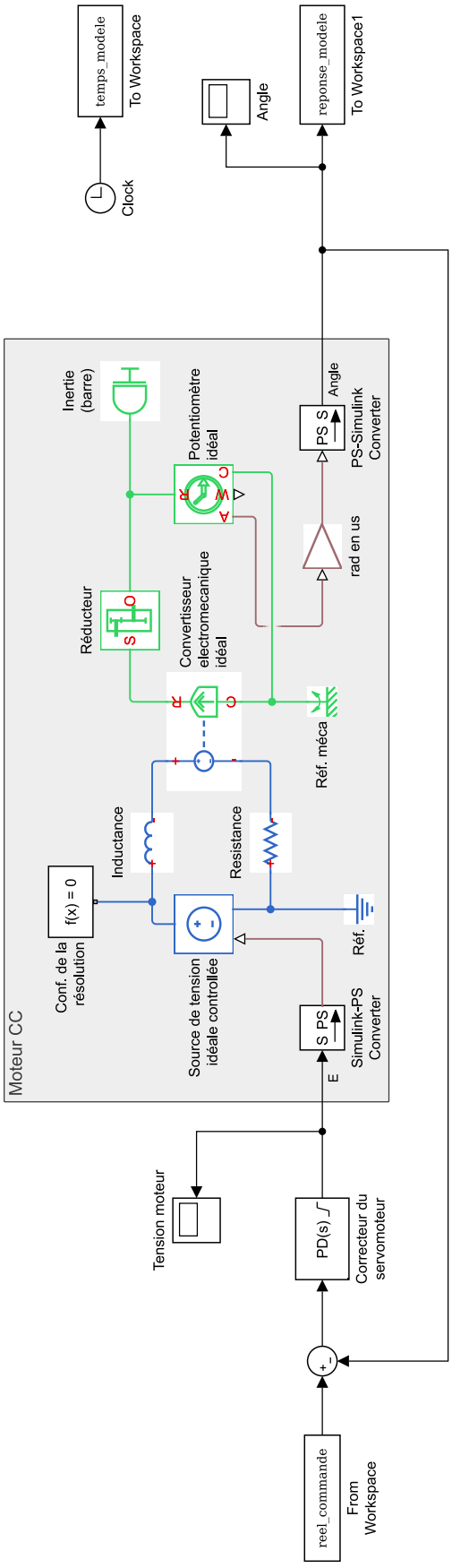
```
% Vitesse de la rampe en entrée (le plus faible possible)
vitesse_rampe = 0.005; % us/ms
n_mesures = 20;

regressions = zeros(n_mesures, 3);
i_mesure = 1;
while i_mesure <= n_mesures
    fprintf('Lancement de la %deme mesure\n', i_mesure);

    % Effectue l'acquisition
    [reel_temps, reel_commande, reel_reponse] = reel_faireAcquisition(vitesse_rampe);

    % Filtrage de la réponse du potentiomètre
    reel_reponse = medfilt1(reel_reponse, 10);

    % Rogne pour ne garder que la rampe
    [reel_temps, reel_commande, reel_reponse] = reel_rognerRampe(reel_temps,
        reel_commande, reel_reponse);

    % Effectue la regression
    [coeff_regression, rcarre] = reel_faireRegression(reel_commande, reel_reponse);

    if rcarre > 0.999
        % Stockage du résultat
        regressions(i_mesure,:) = [coeff_regression, rcarre];

        % Affiche le résultat
        reel_traceRegression(reel_commande, reel_reponse, coeff_regression, rcarre);
        reel_traceCourbe(reel_temps, reel_commande, reel_reponse);

        % Incrémente
        i_mesure = i_mesure + 1;
    end
end
```

### 3.2.2 Test de suivi d'une rampe

```
% Vitesse de la rampe en entrée
vitesse_rampe = 1.5; % us/ms

% Effectue l'acquisition
[reel_commande, reel_reponse] = reel_faireAcquisition(vitesse_rampe);

% Applique la fonction trouvée par regression
reel_reponse_r = reel_reponse./0.3-22.81/0.3;

% Affiche le résultat
reel_traceCourbe(reel_commande, reel_reponse_r);
```

# 4 Simulations MATLAB Simulink

## 4.1 Simulation du servomoteur



## 4.2 Simulation complète du stabilisateur